RECEIVED
CENTRAL FAX CENTER

SEP 07 2005

**Patent**

**Attorney Docket No.: Intel 2207/9800**
**Serial No.: 09/708,722**
**Assignee: Intel Corporation**

## IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

| | | |
|---|---|---|
| APPLICANT | : | Stephen J. JOURDAN et al. |
| SERIAL NO. | : | 09/708,722 |
| FILED | : | November 9, 2000 |
| FOR | : | INSTRUCTION SEGMENT RECORDING SCHEME |
| GROUP ART UNIT | : | 2183 |
| EXAMINER | : | Aimee J. LI |

M/S: APPEAL BRIEFS - PATENTS
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

CERTIFICATION OF FACSIMILE TRANSMISSION
I hereby certify that this Appeal Brief is being facsimile transmitted to the Patent and Trademark Office (Fax No. [illegible]) on September 7, 2005.
[illegible]

## APPEAL BRIEF

Dear Sir:

This brief is in furtherance of the Notice of Appeal, filed in this case on April 7, 2005.

### 1. REAL PARTY IN INTEREST

The real party in interest in this matter is Intel Corporation. (Recorded November 9, 2000, Reel/Frame 011535/0333).

09/08/2005 CCHAU1 00000069 110600 09708722
02 FC:1402 500.00 DA

## 2. RELATED APPEALS AND INTERFERENCES

There are no related appeals.

## 3. STATUS OF THE CLAIMS

Claims 1-19 are pending in this application. Claims 1-19 were rejected under 35 U.S.C. §103(a). This appeal is an appeal from the rejection of claims 1-19.

## 4. STATUS OF AMENDMENTS

Applicants did not make any amendments to the claim subsequent to final rejection. The claims listed on page 1 of the Appendix attached to this Appeal Brief reflect the present status of the claims (including amendments entered after final rejection).

## 5. SUMMARY OF THE CLAIMED SUBJECT MATTER

Embodiments of the present invention provide a recording scheme for instruction segments that store the instruction in reverse program order. By storing the instruction in reverse program order, it becomes easier to extend the instruction segment to include additional instructions. The instruction segments may be extended without having to re-index tag arrays, pointers that associate instruction segments with other instruction segments.

The embodiment of independent claim 1 of the present invention generally describes an instruction segment (see, e.g., Fig. 1, elements 410, 420 – Specification page 5, lines 21-30) comprising a plurality of instructions stored in sequential positions of a cache line (see, e.g., Fig. 3, elements 310.1, 310.2, ... 310.N – page 4, lines 28-31) in reverse program order.

The embodiment of independent claim 5 of the present invention generally describes a segment cache (see, e.g., Fig. 3, element 310 – page 4 line 29) for a front-end system in a processor (see, e.g., Fig. 2, element 200 – page 4 line 3), comprising a plurality of cache entries (see, e.g., Fig. 3, elements 310.1, 310.2, ... 310.N – page 4, lines 28-31) to store instructions of instruction segments (see, e.g., Fig.4, elements 410, 420 – page 5, line 29) in reverse program order.

The embodiment of independent claim 8 of the present invention generally describes a method for storing instruction segments in a processor, comprising: building an instruction segment (see, e.g., Fig. 4, elements 410, 420 – page 5, lines 21-30) based on program flow, and storing instructions of the instruction segment in a cache entry (see, e.g., Fig. 3, elements 310.1, 310.2, ... 310.N – page 4, lines 28-31) in reverse program order.

The embodiment of independent claim 14 of the present invention generally describes a processing engine, comprising: a front end stage to build and store instruction segments (see, e.g., Fig. 4, elements 410, 420 – page 5, lines 21-30), instructions provided therein in reverse program order, and an execution unit in communication with the front end stage (see, e.g., Fig. 2, element 200 – page 4 line 3).

FIG. 2 is a block diagram of a front end processing system 200 according to an embodiment of the present invention.

FIG. 3 is a block diagram of a segment cache 300 according to an embodiment of the present invention. The segment cache 310 may be populated by a plurality of cache lines 310.1, 310.2, ... 310.N, each of which may store an instruction segment. The segment cache 310 may be constructed from any number of cache structures, including for example a set-associative cache or a banked cache among others. According to an embodiment, the segment cache 300 may output a cache line in response to addressing data (not shown) input to the segment cache 300.

FIG. 4 illustrates a relationship between exemplary segment instructions and the manner in which they may be stored in a cache line according to the embodiments of the present invention. In the example of FIG. 4, two different instruction streams are stored in different locations of the instruction cache (FIG. 2, 210). Assume that the first instruction stream extends from a location IP1 to IP2 and the second instruction stream extends from location IP3 to IP4. Assume further that a conditional branch in the first instruction stream at location IP5 may cause program flow to jump to location IP6 in the second instruction stream. For purposes of this example, it also may be assumed that return instructions are located at instruction IP2 and IP4. It further may be assumed that the ISS (FIG. 2, 220) does not store any previously created instruction segments.

During execution, a first segment may begin when program flow advances to location IP1 (as by, for example, a conditional branch). Instructions may be retrieved from the instruction cache 210 until the program flow advances to the conditional branch instruction at location IP5. Assume that the conditional branch is taken, causing program flow to advance to location IP6. In an extended block system, for example, the conditional branch would cause the instruction segment to terminate and a new segment to be created starting at location IP6. The first instruction segment may be stored in a line of the segment cache (say, 310.2 of FIG. 3).

Program flow may advance from location IP6 to the return instruction at location IP4. The return instruction would terminate a second instruction segment 420, causing the ISS (FIG. 2, 220) to store the second instruction segment 420 in another cache line 440. The instructions may be recorded terminal instruction first, then in reverse program order. Thus, the terminal instruction from location IP4 may be stored in a first position 440.1 of the cache line 440. The instructions may be stored in reverse program order in advancing locations of the cache line 440 until the instructions are exhausted. In the example of FIG. 4, the instruction at location IP6 is shown stored in position 440.9 in the cache line 440. The second instruction segment 420 need not occupy the full width of the cache line 440. The first instruction segment 410, when stored in the segment cache 300, also may be stored in reverse program order.

Assume that program flow advances to the instruction at location IP3 at some later time. Instructions may be retrieved from the instruction cache (FIG. 2, 210) until the program flow advances to the return instruction at location IP4. The ISS (FIG. 2, 220) may construct a third instruction segment 430 extending from location IP3 to IP4. Rather than store the third instruction segment 430 in a separate cache line, the ISS 220 instead may extend the second instruction segment 420 to include the additional instructions from the third instruction segment 430. This occurs simply by writing the excess instructions, those from location IP3 to location IP6, at the end of the cache line 440 in reverse instruction order. In an embodiment, if the second instruction segment is subsumed entirely within the third instruction segment, the fastest way of extending the instruction segment is simply to write the third segment 420 into the cache line 440. In this embodiment, the instructions of the second segment are overwritten with identical data.

The second instruction segment 420 is stored in the cache bank 310, the mapping in the segment BPU 270 may reflect the IP of the terminal instruction (IP4) and run length data identifying the number of instructions contained in the second segment 420. When the second and third instruction segments 420, 430 merge, the mapping for the second instruction segment 420 remains valid. Additional information may be stored regarding the third instruction segment

430 to identify the IP of the terminal instruction (again, IP4) and the length of the instruction segment. Thus, the reverse-order-recording scheme provided by the foregoing embodiments facilitates segment extension without requiring a re-indexing of previously stored segments.

## 6. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

A. Are claims 1, 3-8, 12-15 and 17-19 rendered obvious by Patel et al., *Improving Trace Cache Effectiveness with Branch Promotion and Trace Packing* ("Patel"), in view of Johnson, U.S. Patent No. 5,924,092 ("Johnson")?

B. Are claims 2, 9-11 and 16 rejected under 35 U.S.C. 103(a) as being unpatentable over Patel in view of Johnson, in further view of Peled et al., U.S. Patent No. 6,076,144 ("Peled")?

## 7. ARGUMENT

### A. Claims 1, 3-8, 12-15 and 17-19.

The Examiner asserts that it would be obvious to modify the instruction segment of Patel with the teaching of Johnson in order to store instructions of an instruction trace in reverse order "so that the frequently accessed and modified head of the trace will be moved and modified fewer times so that performance is improved." Applicants respectfully disagree. Applicants submit in order to establish *prima facie* obviousness, there must be some suggestion or motivation to modify the reference or combine the reference teachings. For the following reasons, there is no such suggestion or motivation here.

Patel discloses the improvement of fetch rates in trace caches by employing branch promotion and trace packing. Branch promotion removes the overhead resulting from dynamic branch prediction by applying static branch prediction to strongly biased branches. Trace packing packs as many instructions as possible into a pending trace so that more instructions segments may be fetched during a single fetch cycle. However, Patel neither teaches nor suggests that the fetch rates may be improved by reversing the order of the instructions in the traces. Applicants submit there would be no motivation to do so, since reversing instruction order would not appear to improve fetch rates.

By definition, a trace is a sequence of dynamically executed instructions, which may originally reside in non-continuous portions of the program memory, starting with a single entry instruction and ending with multiple exit instructions. For a typical trace, the head of the trace, i.e., the first instruction in a sequence, is followed by the next executable instruction in the sequence, then the next, and so on. If the Examiner's assertions (discussed above) are to be believed, the first instruction is accessed and modified more than the second, third, etc., instructions. This is contrary to the known operation of the typical trace.

Applicants submit it is unclear how accessing the first instruction in a trace more frequently than the second instruction, and accessing the second instruction more frequently than the third, and so on, would improve the performance of a trace (thereby eliminating any motivation to do so). Since the trace defines sequential instructions, which perform a particular operation, accessing the instructions in decreasing frequency would in no way advance the completion of the particular operation. Indeed, such access of the trace would hinder the completion, thereby defeating the purpose of the trace.

Moreover, it is unclear how modifying the first instruction in a trace more frequently than the second instruction, and modifying the second instruction more frequently than the third, and so on, would improve the performance of a trace. Again, since the trace defines sequential instructions, which perform a particular operation, modifying the instructions in decreasing frequency would in no way advance the completion of the particular operation. Indeed, such modification would result in a different operation, thereby, defeating the purpose of the trace.

Therefore, the Examiner's asserted motivation for modifying Patel with Johnson does not apply.

Furthermore, even if the head of the trace could be more frequently accessed and modified, the Examiner has provided no explanation of how such would improve the fetch rates of the trace cache of Patel.

As stated previously, there is no motivation to reverse the instructions in a Patel trace. Patel discloses using branch promotion to improve cache fetch rates. The purpose of branch promotion is to reduce the dynamic branching of strongly biased traces by applying static branches (or predictions). Applicants submit reversing the instructions so that the first instruction is listed last in the trace does not improve the branch promotion technique. Reversing the instructions does not reduce the dynamic branching of strongly biased traces. Moreover, it does not improve the fetch rates. As such, there is no reasons a person of ordinary skill in the art would be motivated to reverse the instructions in a trace, while using branch promotion, to improve fetch rates.

Patel also discloses using trace packing to improve cache fetch rates. The purpose of trace packing is to increase the number of instructions fetched per fetch cycle. However, Applicants submit reversing the instructions so that the first instruction is listed last in the trace does not improve the trace packing technique. Moreover, such does not appear to improve the fetch rates. As such, a person of ordinary skill in the art would not be motivated to reverse the instructions in a trace, while using trace packing, to improve fetch rates.

Accordingly, the Examiner has failed to establish a *prima facie* case of obviousness over Patel in view of Johnson.

**B. Claims 2, 9-11 and 16**

The deficiencies are not corrected by Peleg. Peleg discloses a cache organized around trace segments of the running prgrams rather than an organization based on memory addresses. However, Peleg fails to provide any motivation for modifying Patel with Johnson. Moreover, there is no motivation disclosed to modify Patel with Johnson and Peled to arrive at the claimed invention. Accordingly, the Examiner has failed to establish a *prima facie* case of obviousness over Patel in view of Johnson in further view of Peled.

For at least these reasons, the Claims 1-19 are believed to be patentable over the cited references, individually and in combination. Withdrawal of the rejections is, therefore, respectfully requested.

Appellants therefore respectfully request that the Board of Patent Appeals and Interferences reverse the Examiner's decision rejecting claims 1-19 and direct the Examiner to pass the case to issue.

The Examiner is hereby authorized to charge the appeal brief fee of **$500.00** and any additional fees which may be necessary for consideration of this paper to Kenyon & Kenyon Deposit Account No. **11-0600**.

Respectfully submitted,

KENYON & KENYON

Date: September 7, 2005

By: [illegible]
Sumit Bhattacharya
(Reg. No. 51,469)

KENYON & KENYON
333 West San Carlos St., Suite 600
San Jose, CA 95110
Telephone: (408) 975-7500
Facsimile: (408) 975-7501

## APPENDIX

(Brief of Appellants Stephan J. Jourdan, et al.
U.S. Patent Application Serial No. 09/708,722)

## 8. CLAIMS ON APPEAL

1. (Original) An instruction segment comprising a plurality of instructions stored in sequential positions of a cache line in reverse program order.

2. (Original) The instruction segment of claim 1, wherein the instruction segment is an extended block.

3. (Original) The instruction segment of claim 1, wherein the instruction segment is a trace.

4. (Original) The instruction segment of claim 1, wherein the instruction segment is a basic block.

5. (Previously presented) A segment cache for a front-end system in a processor, comprising a plurality of cache entries to store instructions of instruction segments in reverse program order.

6. (Previously presented) Apparatus comprising:

an instruction cache system,

an instruction segment system, comprising:

a fill unit provided in communication with the instruction cache system,

the segment cache of claim 5 included therein, and

a selector coupled to an output of the instruction cache system and to an output of the segment cache.

7. (Previously presented) Apparatus of claim 6, wherein the instruction segment system further comprises a segment predictor provided in communication with the segment cache.

8. (Previously presented) A method for storing instruction segments in a processor, comprising:

building an instruction segment based on program flow, and

storing instructions of the instruction segment in a cache entry in reverse program order.

9. (Original) The method of claim 8, further comprising:

building a second instruction segment based on program flow, and

if the first and second instruction segments overlap, extending the first instruction segment to include non-overlapping instructions from the second instruction segment.

10. (Original) The method of claim 9, wherein the extending comprises storing the non-overlapping instructions in the cache in reverse program order in successive cache positions adjacent to the instructions from the first instruction segment.

11. (Original) The method of claim 8, wherein the instruction segment is an extended block.

12. (Original) The method of claim 8, wherein the instruction segment is a trace.

13. (Original) The method of claim 8, wherein the instruction segment is a basic block.

14. (Previously presented) A processing engine, comprising:

a front end stage to build and store instruction segments, instructions provided therein in reverse program order, and

an execution unit in communication with the front end stage.

15. (Previously presented) The processing engine of claim 14, wherein the front-end stage comprises:

an instruction cache system,

an instruction segment system, comprising:

a fill unit provided in communication with the instruction cache system,

a segment cache, and

a selector coupled to an output of the instruction cache system and to an output of the segment cache.

16. (Previously presented) The processing engine of claim 15, wherein the instruction segments are extended blocks.

17. (Previously presented) The processing engine of claim 15, wherein the instruction segments are traces.

18. (Previously presented) The processing engine of claim 15, wherein the instruction segments are basic blocks.

19. (Previously presented) The processing engine of claim 15, wherein the instruction segment cache system further comprises a segment predictor provided in communication with the segment cache.

## 9. EVIDENCE APPENDIX

No further evidence has been submitted with this Appeal Brief.

## 10. RELATED PROCEEDINGS APPENDIX

Per Section 2 above, there are no related proceedings to the present Appeal.